

# METHOD AND SYSTEM FOR RESOLVING MEMORY LEAKS AND RELEASING OBSOLETE RESOURCES FROM USER SESSION DATA

## Background of the Invention

### 1. Field of the Invention

5

The present invention relates to the art of information processing. It finds particular application in Java programming for resolving memory leaks due to failures of the automatic memory management system, and will be described with particular reference thereto. However, the present invention is useful in many other programming environments that employ automatic memory management, such as the C# programming environment.

10

### 2. Description of Related Art

A feature of Java, C#, and certain other programming environments is automatic memory management, in which one or more garbage collector algorithms are provided to automatically remove objects that are no longer in use. Programming environments with automatic memory management are less susceptible to memory leaks introduced by failure of programmers to remove unneeded objects in a timely manner.

15

Even with automatic memory management, however, memory leaks can arise due to various failures of the automatic memory management. In one common situation, a programmer fails to properly dereference one or more system resource

20

references, such as file handles, database connections, sockets, threads, or the like. This leads to objects containing obsolete references that inhibit or prevent their removal by the garbage collector. Moreover, in addition to not collecting the object containing the obsolete references, the garbage collector will be inhibited or prevented from collecting  
5 any objects referenced by the object containing these obsolete references. In this way, a few obsolete references can lead to a large number of uncollected objects.

Memory leakage is a general problem that can affect any type of computer program; however, memory leaks are particularly problematic in certain programming applications where such leaks can accumulate over time to reduce the available free  
10 memory. For example, in a server application, a separate user session is opened for each user who accesses the server. The user session expires when the user logs out, closes the browser, or otherwise disengages from the server. When the session expires, the user session objects should be collected by the garbage collector to free up memory. Objects containing obsolete references may not be collected, however, and can accumulate over  
15 time as users successively log onto and disconnect from the server. Eventually, such accumulated memory leaks lead to a server crash as the available memory becomes filled.

The present invention contemplates an improved method and apparatus which overcomes these limitations and others.

### **Summary of the Invention**

20 In accordance with one aspect of the invention, a method of releasing resources of a user session is provided. The user session operates in a software environment that includes an automatic memory management algorithm. An impending

execution of the automatic memory management algorithm is detected. Responsive to the detecting, an object of the user session is accessed. One or more external resource references of said object are identified. Said one or more external resource references are released. The accessing, identifying, and releasing are repeated for each object of the user  
5 session.

In accordance with another aspect of the invention, an article of manufacture is disclosed, comprising a program storage medium readable by a computer and embodying one or more instructions executable by the computer to perform a method for preparing a user session for expiration. An impending expiration of the user session is  
10 detected. An object graph corresponding to the user session is traversed to locate user session objects. For each object located in the traversing, allocated resources of the object are identified. For each identified allocated resource, said allocated resource is deallocated.

In accordance with yet another aspect of the invention, a system is  
15 disclosed. A software program is configured to initiate, process, and terminate user sessions. A resource deallocation module is linked to the software program to deallocate allocated external resources of each object of a user session responsive to an impending termination of said user session. An automatic memory management module is invoked subsequent to the deallocation performed by the resource deallocation module.

20 Numerous advantages and benefits of the invention will become apparent to those of ordinary skill in the art upon reading and understanding this specification.

### **Brief Description of the Drawing**

The invention may take form in various components and arrangements of components, and in various process operations and arrangements of process operations. The drawing is only for the purposes of illustrating preferred embodiments and is not to  
5 be construed as limiting the invention.

FIGURE 1 shows a block diagram of a server application operating in a Java environment that includes an automatic memory management module and a resource deallocation module.

### **Detailed Description of the Preferred Embodiments**

10 With reference to FIGURE 1, a Java-based server executes within a Java virtual machine **10**, which is indicated by a dashed box in FIGURE 1. The Java virtual machine **10** executes Java programs in bytecode format. The Java virtual machine **10** is suitably a pure interpreter, a just-in-time compiler, or the like. The server is implemented as a server program **12** that is stored in bytecode format on a magnetic disk, optical disk,  
15 or other suitable digital data storage medium. Typically, the server program **12** or a selected portion thereof is loaded into random access memory (RAM) for execution by the Java virtual machine **10**.

In the course of server operation, a first user **20** logs onto the server and a first user session **22** is created. Similarly, a second user **24** logs onto the server and a  
20 second user session **26** is created. Other users can similarly log onto the server, and a separate user session is created for each such user.

During execution of each user session **22, 26**, the server program **12** typically creates user session objects that perform various tasks requested by the user **20, 24**. These user session objects, in turn, may create additional user session objects that are referenced by the generating user session objects. Each user session **22, 26** preferably  
5 includes an object graph **30, 32** or other structure that keeps track of the objects of the corresponding user session **22, 26** and that keeps track of interrelationships or references between the session objects.

Moreover, the various user session objects may access system resources such as files, databases, other networked servers, and the like. When such system  
10 resources are accessed, the user session object creates external resource references such as file handles, database connections, sockets, threads, and the like, that are associated with or contained in the user session object.

Over time, certain user session objects which were created to perform certain tasks may have completed those tasks and no longer be needed. That is, once a  
15 given user session object completes the task for which it was created, that user session object is no longer needed. To free up memory occupied by user session objects that are no longer needed, an automatic memory management system is provided. Specifically, a garbage collector **40** executes one or more garbage collection algorithms with respect to one or more selected user sessions. The garbage collection algorithm or algorithms  
20 determine whether each user session object is still needed, and removes user session objects which are no longer needed.

In a typical approach, the garbage collector **40** examines the object graph **30, 32** of the user session being processed. User session objects which are not referenced

by any other user session objects and which do not include references to external resources are deemed to be unneeded, and are removed. Typically, the garbage collector 40 is an integral part of the Java virtual machine 10 and is available to automatically manage memory usage of any Java program executing on the Java virtual machine 10. It is also contemplated, however, for the garbage collector 40 to be an add-on component 5 respective to the Java virtual machine 10. For example, the garbage collector may be a Java program written specifically for managing memory of the server program 12.

The garbage collector 40 may fail to collect certain user session objects. For example, some user session objects created by the server program 12 may fail to properly dereference one or more system resource references after the user session object 10 is finished accessing the corresponding system resource. Such obsolete references can include, for example, file handles, database connections, sockets, threads, or the like. These obsolete references are detected by the garbage collector 40 and misinterpreted as indicating that the user session object is still needed. Thus, the garbage collector 40 fails 15 to collect the user session object. An example of such a potential problem is given by the following programming example:

```
File file = new File("sample.txt");
FileReader filerdr = new FileReader(file)
20  BufferedReader fr = new BufferedReader(filerdr);
    session.putValue("filehandler", filerdr);
    session.putValue("bufferfilehandler", fr);
    .
    .
25  .
    {
        BufferedReader fr = (BufferedReader)
            session.getValue("bufferfilehandler");
```

```

String finalstr = "";
String str = "";

5      while((str = fr.readLine()) != null)
      {
          finalstr = finalstr + str;
      }

10     // Nulling out the BufferedReader reference

      fr = null;
      filerdr = null;

15  } catch(IOException iox)
      {

      }

```

20 In typical Java implementations, merely nulling out the file reference `fr` and file reader stream reference `filerdr` does not deallocate the external file resource. In this case, the user session object containing the above portion of code retains obsolete references `fr` and `filerdr` which prevent the garbage collector 40 from removing the user session object and freeing the corresponding memory. Moreover, in addition to not collecting the user

25 session object containing the above portion of code, the garbage collector 40 will also be unable to collect any user session objects referenced by the user session object containing the above portion of code.

These retained obsolete user session objects are memory leaks. That is, they represent portions of memory that are occupied by unused user session objects, and

30 those portions of memory are therefore unavailable for other uses. It will be appreciated that these memory leaks are not typically a serious problem for a single user session. For

example, a memory leak or even several memory leaks in the user session **22** are unlikely to grow sufficiently large to create a server failure by itself.

Server failure is typically caused by accumulation of memory leaks over time. As each user session is created, generates memory leaks, and then expires, it leaves  
5 behind obsolete user session objects that are not recovered by the garbage collector **40**. Over time, the accumulated leakage of many such user sessions accumulates until a substantial amount of memory is occupied by such leaks. The reduced available memory can result in server slowdown, a server crash, or other detrimental server behavior.

When a user session is about to expire, the garbage collector **40** is invoked  
10 with respect to that user session. In the absence of memory leaks due to obsolete references, the invocation of the garbage collector **40** just prior to expiration of the user session results in all user session objects being collected (since no user session object should be needed at that point), so that when the user session expires the memory formerly occupied by that user session is freed up.

15 To address the problem of memory leaks due to obsolete references, a resource deallocation module **50** is executed prior to the garbage collection to deallocate any remaining allocated external resources of the about-to-expire user session. The resource deallocation module **50** accesses the object graph of the user session, traverses the object graph, dereferences user session objects by applying methods to remove  
20 references by a set of rules for a given user session object, and also deallocates any resources owned by the user session object by applying methods to release resources by a set of rules for a given object.

In a preferred embodiment for Java applications, the resource deallocation module **50** is implemented as a Listener method belonging to a Java MyListener class. The listener is registered with each user session, for example by assigning the Listener method to a session attribute **60, 62** using the following Java code snippet:

5

```
session.Attribute("Deallocator", instanceof.Listener)
```

which assigns an instance of the Listener embodying the resource deallocation module **50** to a session attribute **60, 62** identified as "Deallocator". In this way, just before the user session is about to expire and be destroyed, the session Listener instance is notified and operates to gracefully release resources held by user session objects of the about-to-expire user session.

A suitable method executed by the resource deallocation module **50** is set forth in the following programming example:

15

```
public void startReleaseResources()
{
    // An enumeration of all the objects in the session is obtained
    String[] enum = session.getValueNames();
    int size = enum.length;

    for(int i=0;i<size;i++)
    {
        String name = enum[i];

        // Retrieve an object from the session.

        Object obj = session.getValue(name);

        // Inspect the object type.
        // Different types have different dereferencing mechanisms.
```

20

25

30

```

        obj.releaseResources();
    } //end-for
} // end-method
}

```

5

where the comments "// Inspect the object type" And "// Different types have different dereferencing mechanisms" are replaced by programming language implementation-specific and optionally object-specific code that identifies the object type and the release mechanism for references therein. For example, an identified file resource is properly released by closing the file resource (and not by merely nulling out the associated references). Other allocated resources such as database connections, sockets, threads, and the like are suitably deallocated using other specific deallocation mechanisms.

In the above programming example, string array "enum[]" is loaded with an enumeration of identifiers of objects of the user session. The "for{" loop employs integer index "i" to cycle through the user session objects identified by "enum[i]". For each object, the obj.releaseResources() method identifies and releases any remaining allocated resources of that user session object. The obj.releaseResources() method is optionally replaced by one or more suitable release methods tailored for a particular programming language, allocated resource, object, or other particulars.

After the resource deallocation module **50** has gone through the enumerated objects of the user session and completed the task of releasing any remaining references within each object, the garbage collector **40** is invoked to perform the usual garbage collection for the about-to-expire user session. Since any obsolete references of the user session objects have been removed by the resource deallocation module **50**, the

25

garbage collector **40** frees the user session objects without leaving memory leaks in the form of unfreed user session objects left over due to improper hanging references. Rather than invoking the garbage collector **40** responsive to an impending termination of a user session, the garbage collector **40** can be executed at the server program **12** level for multiple sessions. Still further, the garbage collector **40** can be executed at the level of the  
5 Java virtual machine **10** or at the operating system level for multiple server applications.

The Java server application described with reference to FIGURE 1 is exemplary only. Those skilled in the art can readily adapt the described resource deallocation module **50** to other applications besides server applications. Moreover, the  
10 resource deallocation module **50** is readily employed in other programming environments that include automatic memory management, such as C#. Similarly to Java, C# executes intermediate language (IL) code and supports automatic garbage collection. Moreover, the resource deallocation module **50** is not limited to interpreter or just-in-time compiler environments such as Java and C# virtual machines that execute machine-independent  
15 code. The server program **12**, garbage collector **40**, and resource deallocation module **50** can be embodied as a fully compiled program or collection of programs created using Java, C#, or another compilable programming language and fully compiled into native machine code prior to execution.

Although the resource deallocation module **50** is shown as a separate  
20 module in FIGURE 1, it is also contemplated to incorporate some or all of the functionality of the resource deallocation module **50** into the garbage collector **40** or into the server program **12**. The resource deallocation module **50** can be an integral component of the java virtual machine **10** or can be an add-on utility program.

Still further, the functionality of the resource deallocation module **50** optionally is extended to address other issues that inhibit or prevent the garbage collector **40** from freeing all user session objects. For example, the resource deallocation module **50** optionally addresses certain configurations of circular object references that may be problematic for the garbage collector **40**. The resource deallocation module **50** can address such object cycles, for example, by removing one or more object references from one or more of the objects of the cycle to break the reference circularity. With the problematic object cycle broken, the garbage collector **40** can readily collect the objects.

In the described augmented memory management method, the resource deallocation module **50** is preferably executed just prior to expiration of the user session but before the final garbage collection of that user session. In certain memory-intensive applications, however, it may be typical for the garbage collector to be invoked at other times besides just before user session expiration. For example, the garbage collector may be invoked whenever the free memory available for the application drops below a selected threshold. In such situations, the resource deallocation module **50** is readily modified to support such intermediate garbage collecting. For example, execution of the resource deallocation module can be triggered by an available free memory threshold that is above the aforementioned memory threshold for invoking the garbage collector. This ensures that resource deallocation is performed prior to the intermediate garbage collection. In the case of intermediate garbage collection performed while the user session is still in progress, some external references may still be active and should be retained. Hence, a less aggressive deallocation than the `obj.releaseResources()` method should be employed. For example, the substitute method optionally releases only null file

references and other null external resource references, which are likely to have resulted from improper and failed release the resource by nulling rather than by using a deallocation method appropriate for deallocating that resource.

Still yet further, the resource deallocation method described herein is not  
5 limited to applications that employ multiple user sessions. For example, similar memory leak problems can arise on single-user personal computers, personal digital assistants, and other digital data processing systems in which various programs are opened, executed, and closed under an operating system such as Windows, PalmOS, MacOS, UNIX, LINUX, or the like. If the operating system employs an automatic memory  
10 management system, the resource deallocation module 50 described herein is readily adapted to augment the automatic memory management system to prevent accumulation of obsolete objects left over after various application programs are closed. In this adaptation of the resource deallocation module 50, the various user application programs correspond to user sessions 22, 26, and the operating system corresponds to the server  
15 program 12.

The invention has been described with reference to the preferred embodiments. Obviously, modifications and alterations will occur to others upon reading and understanding the preceding detailed description. It is intended that the invention be construed as including all such modifications and alterations insofar as they come within  
20 the scope of the appended claims or the equivalents thereof.